First, we import a number of libraries that will be used in the analysis

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import os

The os library is then used to loop through all the files in the /kaggle/input directory and print the file names. This step is not necessary for the analysis and can be skipped.

Next, we load the heart.csv dataset into a Pandas dataframe called df and display the first 5 rows using the head() method:

```
df = pd.read_csv('heart.csv')
```

df.head()

We can get some information about the data using the info() method:

df.info()

We then check for missing values in the data using the isna() method and the sum() method:

df.isna().sum()

Next, we create a statistical summary of the data using the describe() method and transpose the results using the transpose() method:

df.describe().transpose()

We can also create a pairplot using the pairplot() method from the seaborn library, setting the hue parameter to HeartDisease:

sns.pairplot(df,hue='HeartDisease')

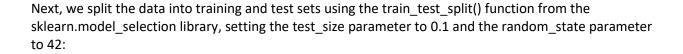
We can create a heatmap of correlations between features using the heatmap() method from the seaborn library and setting the figsize parameter to a tuple of (6,4) and the dpi parameter to 150:

```
plt.figure(figsize=(6,4),dpi=150)
```

sns.heatmap(df.corr(),cmap='viridis',annot=True)

We can find the top 5 correlated features with the target variable by taking the absolute value of the correlations using the abs() method, sorting the values using the sort\_values() method, and selecting the last 6 values using the tail() method:

np.abs(df.corr()['HeartDisease']).sort values().tail(6)



from sklearn.model\_selection import train\_test\_split X = df.drop('HeartDisease',axis=1) y = df['HeartDisease'] X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.1, random\_state=42) We can check the shape of the training and test sets using the shape attribute: X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape contniue further i will only type c for continue Certainly, here is the rest of the code: We can check the balance of the label data in the training set by using the value\_counts() method and plotting the results using the countplot() function from the seaborn library: y\_train.value\_counts() sns.countplot(x=y\_train)

We can calculate the percentage of the minority class in the training set by dividing the number of

minority class samples by the total number of samples:

372/(454+372)

Next, we perform feature engineering by creating a ColumnTransformer object to scale the numeric features and one-hot encode the categorical features. We create lists of the numeric and categorical columns and instantiate a StandardScaler object and a OneHotEncoder object, setting the drop parameter of the OneHotEncoder to 'first' to drop one of the encoded columns:

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.compose import ColumnTransformer

We then create a base model of logistic regression by instantiating a LogisticRegression object and setting the max\_iter parameter to 1000:

from sklearn.linear model import LogisticRegression

```
base_model = LogisticRegression(max_iter=1000)
```

We create a pipeline by instantiating a Pipeline object and passing it a list of tuples containing the steps in the pipeline, in this case the preprocessor and the base model:

from sklearn.pipeline import Pipeline

```
pipe = Pipeline([('preprocessor',preprocessor),('base model',base model)])
```

We perform a grid search using the GridSearchCV function from the sklearn.model\_selection library, passing it the pipeline and a dictionary of parameters to test. We set the scoring parameter to 'accuracy' and the cv parameter to 5:

from sklearn.model\_selection import GridSearchCV

```
parameters = ({'base_model__C': [0.001,0.01,0.1,1,10],
        'base_model__penalty': ['l1','l2','elasticnet'],
        'base_model__solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
       })
grid = GridSearchCV(estimator=pipe,param_grid=parameters,scoring='accuracy',cv=5)
grid.fit(X_train,y_train)
We can find the best estimator and best parameters using the best_estimator_ and Certainly, here is the
rest of the code:
We can find the best estimator and best parameters using the best_estimator_ an best_params_
attributes of the grid object:
best_model = grid.best_estimator_
best_params = grid.best_params_
print(best_model)
print(best_params)
We can calculate the accuracy of the model on the test set using the score() method:
accuracy = best_model.score(X_test,y_test)
print(accuracy)
We can also make predictions using the predict() method and calculate the confusion matrix using the
confusion_matrix() function from the sklearn.metrics library:
from sklearn.metrics import confusion matrix
y_pred = best_model.predict(X_test)
confusion_matrix = confusion_matrix(y_test,y_pred)
print(confusion_matrix)
```

Finally, we can plot the confusion matrix using the heatmap() function from the seaborn library and setting the annot parameter to True:

```
plt.figure(figsize=(4,4))
sns.heatmap(confusion_matrix,annot=True,cmap='Blues',fmt='d')
plt.ylabel('True label')
plt.xlabel('Predicted label')
This completes the analysis of the heart disease dataset using logistic regression.
```